[0537] It also becomes apparent that two object connections are required for each component. To reduce the number of object connections (allowing greater flexibility), and to add more flexibility to modify events that are fired from the model before they reach the view controller an additional modification is made. The view sends its events only to the view controller and the model sends its events only to the model controller. The need for the input controller goes away as the responsibility is placed in the "view controller" hierarchy. The model sends its events to the "model controller" instead. The additional functionality, which otherwise would be gained by adding an "input controller" equivalent to the model side, is added to the "model controller".

[0538] Finally, it becomes apparent that the input events understood by a "view in controller" and the events generated and sent out through the "view out controller" are related and may be managed by the same component. The same is true for the "model in controller" and the "model out controller". As a side effect of this consolidation of controllers, events no longer flow from the model to the view controller or from the view to the model controller, instead all events flow between the view controller and the model controller. This allows the insertion of "forward" components between the model controller and view controller, which simply convert an event for transport over a network and convert it back once it arrives on the other side.

[0539] This results in a HMVC design where from the model side event flow from the model to several submodels. Events from the several submodels flow to a respective subcontroller for each submodel. Each of the subcontrollers transfers events to the model controller. The model controller then transfers events to the model. Events transfer from the Model Controller to the Forward to View and events transfer from the Forward to View to the Model Controller.

[0540] In the view side of the HMVC design, events transfer from the view to several subviews that in turn transfer events to respective subcontrollers. The subcontrollers transfer events to the view controller. And events transfer back and forth between the view controller and the forward to model. Also events transfer to the view controller from the view.

[0541] The final result is an enhanced version of MVC (HMVC 4.6.2) that uses metadata in order to reduce the number of events from several hundred different types of events to a set of twelve models. Additional models can be added to the HMVC pattern as necessary, but the design will continue to keep the number of models low. HMVC 4.6.2 can programmatically check whether a model will correctly work with a view, which is a task ordinarily left to error-prone human programming. HMVC 4.6.2 is component oriented, supporting increased code reuse while simultaneously decreasing the number of direct object connections to allow greater flexibility.

[0542] As a result of the new design, "forward" components can be inserted, adding support for transport over networks without modifying any code in the model, view, model controller, or view controller. As a result of the separation of view logic from view controller logic, a sophisticated view controller can dynamically construct an appropriate view for presenting data for any model based on the model controller's metadata. As a result of the separation of model control logic from the model logic, a sophisticated model controller can support actions (business logic) added dynamically through the CI Engine and which define its data (metadata, enable state, values) through the metadata retrieved for the model the CI Engine is given. Without writing any code, this model controller can be combined with the dynamic view controller to automatically create a view for a model. Using the component integration engine allows the configuration of a forward to support a network distribution, and automatically inherits scalability, security, and database persistence from the component integration engine. Since metadata can be used to describe database records as a model object, the HMVC 4.6.2 model results in a network or web-enabled 3-tier client-server application with dynamically configurable functionality, with automatically created views, automatic view controller logic, and automatic model controller logic without programming a single line of source code.

[0543] The use of metadata in events to describe the types of available application functions and data, the definitions of these units, the current state of these units, and the current value of these units allows greater reuse of smaller user interface components. Metadata in events allows a view controller to use the metadata to assemble the correct individual components necessary for display. The metadata used in the events is the descriptors used in the meta-implementation layer.

[0544] The HMVC4.6.2 design of the present invention makes use of metadata for defining the events that will be exchanged rather than relying on the programmer to know what events will be passed. Additionally, it provides new mechanisms for coordinating the appropriate view to a model by comparing metadata definitions.

EXAMPLE 1

[0545] The interrelationships between concepts of the present invention may be illustrated by describing the problem domain of pet store management in a hypothetical pet store. A pet store may contain many animals: bunnies, birds, cats, dogs, and fish. Details related to circulatory system processes, eyesight acuity, and fastest recorded land-speed are not related to the problem domain and are abstracted away from the software model. The types of animals suggest the creation of a software model for Bird, Bunny, Cat, Dog, and Fish. Common features between the various types of animals could be grouped into a base model, animal. Animal is an abstraction of Bird, Bunny, Cat, Dog, and Fish because it does not include details specific to these animal types but includes only features common to all these animal types. Dog is a sub-model of Animal that is generally loyal, may slobber a bit, and enjoys chasing cats.

[0546] A pet store may also require models related to pet food inventory management. Models related to food inventory are still related to the pet store problem domain, but are not logically similar to software models for the animals. Therefore the Animal, Bird, Bunny, Cat, Dog, and Fish models could be grouped into a package named "animals". The software models related to food management could be grouped into a package named "inventory". Both of these packages would be contained in a parent package named "pet store".